OpenPrefetch

Let There Be Industry-Competitive Prefetching in RISC-V Processors
(in-progress)

Bowen Huang, Zihao Yu, Zhigang Liu, Chuanqi Zhang, Sa Wang, Yungang Bao

Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS)
Agenda

Review of existed hardware prefetchers
- Questionable simulation prevails
- No RTL-level implementation & evaluation
- Industry interest is limited

Why we choose RISC-V
- Opensource, fairly good baseline
- RISC-V implementation also lacks hardware prefetcher
- Interests aligned

Current Progress
- Baseline Implementation
- Adding L2 cache to RISC-V processor
Review

15 hardware prefetcher designs published on top4 conferences during the last 8 years

None of it has provided RTL-level implementation or evaluation

We can (only) confirm that BOP[7], SPP[8] and TPC[10] have impact on industry designs so far

[1] Linearizing Irregular Memory Accesses for Improved Correlated Prefetching. MICRO 2013.
[9] Domino Temporal Data Prefetcher. HPCA 2018
[10] Division of Labor - A More Effective Approach to Prefetching. ISCA 2018
Review

Inaccurate simulation

Industry has began to use exclusive cache design (Intel Skylake-EX / ARM Cortex-A55/A75)

Non-inclusive cache vs Exclusive Cache

<table>
<thead>
<tr>
<th>Application</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>bwaves</td>
<td>1.05</td>
</tr>
<tr>
<td>gcc</td>
<td>1.1</td>
</tr>
<tr>
<td>GemsFDTD</td>
<td>1.15</td>
</tr>
<tr>
<td>ibm</td>
<td>1.2</td>
</tr>
<tr>
<td>leslie3d</td>
<td>1.25</td>
</tr>
<tr>
<td>libquantum</td>
<td>1.3</td>
</tr>
<tr>
<td>mcf</td>
<td>1.35</td>
</tr>
<tr>
<td>milc</td>
<td>1.4</td>
</tr>
<tr>
<td>omnetpp</td>
<td>1.05</td>
</tr>
<tr>
<td>soplex</td>
<td>1.1</td>
</tr>
<tr>
<td>wrf</td>
<td>1.05</td>
</tr>
<tr>
<td>zeusmp</td>
<td>1.1</td>
</tr>
<tr>
<td>astar</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Review

**Inconsistent simulation**

- **5% - 10%** speedup with ~100 cycle DRAM latency
- **10% - 20%** speedup with ~290 cycle DRAM latency
- Different DRAM model can reverse the performance rank of existed prefetchers

**An example from calibrated simulator**

<table>
<thead>
<tr>
<th></th>
<th>LLC MPKI</th>
<th>IPC</th>
<th>Useless Prefetch</th>
<th>DRAM read Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>No LLC prefetcher</td>
<td>21.90</td>
<td>0.681</td>
<td>N/A</td>
<td>299 cycles</td>
</tr>
<tr>
<td>SPP</td>
<td>3.55</td>
<td>0.704</td>
<td>&lt;0.01%</td>
<td>286 cycles</td>
</tr>
<tr>
<td>Testing Prefetcher</td>
<td>1.81</td>
<td>0.662</td>
<td>&lt;0.01%</td>
<td>450 cycles</td>
</tr>
</tbody>
</table>

**MPKI can be utterly outweighed by the DRAM**

**Our industry research partner confirmed this phenomenon**
Review

We need one step further ... A new platform for architecture research

- Simulations are inaccurate
- Simulation results are inconsistent across papers
- Implementation difficulties are unexplored

Why RISC-V?

- Opensourced
- Linux-compatible
- Highest performance core we can find so far\(^1\)
- We need RISC-V, and vice versa.

\(^1\) OpenPiton: An Open Source Manycore Research Framework. ASPLOS 2016
Our interests are aligned

RISC-V needs to improve architectural performance

Although BOOMv2\(^1\) is reported to be faster than ARM Cortex A9 ...

Architectural performance bar of ARM's lineup\(^2\)[\(^3\)] is rising \(~25\%\) per generation

---

\[\text{Estimated Relative IPC}\]

---

\(^1\) BOOMv2: an open-source out-of-order RISC-V core, RISC-V Workshop 2017
\(^2\) The final ISA showdown: Is ARM, x86, or MIPS intrinsically more power efficient? ExtremeTech 2014
Our interests are aligned

RISC-V community has already began to pay attention to prefetching

Discussion can be found in “[hw-dev] Data Cache Prefetching for Rocket (and Boom)”
Prefetching Performance on Intel Haswell Xeon

- **Intel Xeon E5-2658v3 (Haswell, 2.2GHz, L3 Cache 30MB)**
- Running on a separated 3MB capacity of LLC by CAT, which is on par with current mobile SoC
- **gcc 4.8.5 -O3, linux kernel 3.10, 4KB page**
- **20.8% on average, 117% for libquantum**

**Speedup gained by L1/L2 prefetching**

<table>
<thead>
<tr>
<th>Application</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>bzip2</td>
<td>1.00</td>
</tr>
<tr>
<td>gcc</td>
<td>1.10</td>
</tr>
<tr>
<td>mcf</td>
<td>1.20</td>
</tr>
<tr>
<td>gobmk</td>
<td>1.30</td>
</tr>
<tr>
<td>hmmer</td>
<td>1.40</td>
</tr>
<tr>
<td>sjeng</td>
<td>1.50</td>
</tr>
<tr>
<td>libquantum</td>
<td>2.00</td>
</tr>
<tr>
<td>h264ref</td>
<td>1.10</td>
</tr>
<tr>
<td>omnetpp</td>
<td>1.20</td>
</tr>
<tr>
<td>astar</td>
<td>1.30</td>
</tr>
</tbody>
</table>
L1 Prefetcher Prelim. Evaluation

Simulation result on a 4-issue BOOM model, tested with SPECCPU 2006

- **Next Line Prefetcher** ≈ +20% IPC
- **Instruction-Pointer Based Stride Prefetcher** ≈ +30% IPC
How can it be industry-competitive

Exploring more

- Connect prefetcher with TLB
  - +13% perf on milc vs +60% perf on milc (with paging info)
  - Our industry partners won’t try this due to TLB bandwidth

- Hybrid Prefetcher
  - ARM hasn’t adopted this, but we do know somebody implement this already

Exploring faster

- Real case from our industry partner
  - 90 man-months to build a semi-new simulator
  - A full day to run ~1B instructions

- Development with Chisel and FPGA emulation are much faster
But ...

The Official RISC-V implementation lacks L2 cache
- Prefetching can't be too aggressive, to avoid L1 cache pollution
- L1 cache MSHR and input/output buffer size are limited

Forked implementations are using outdated L2 cache
- Both LowRISC / Labeled RISC-V falls into this category

How can we solve that?
Our interests are aligned

The official opensourced RISC-V implementation / SIFIVE products

- BOOM / Rocket both have parameterized L1 cache
- SIFIVE implement L2 cache for its commercial products\(^1\)

- 16-way set-associative
- Multi-banked design
- Scratchpad
- Waymasking and locking
- Cache-coherent
- ECC

\(^1\) [https://www.sifive.com/products/hifive-unleashed/](https://www.sifive.com/products/hifive-unleashed/)

\(^2\) [https://static.dev.sifive.com/FU540-C000-v1.0.pdf](https://static.dev.sifive.com/FU540-C000-v1.0.pdf)
Our interests are aligned

We (Labeled RISC-V[1] team) need high-performance L2 cache

- Our research heavily lies on shared cache & memory controller
- Prefetch into L2 cache / Multi-layer prefetchers
- (auto) cache capacity allocation, (auto) memory bandwidth allocation

A New L2 Cache Design

Highlights

- Highly parameterized (capacity, associativity, buffer queue size, packet-based flow ...)
- Heavy storage structures are implemented with single-port cell
- Multi-banked, 2-cycle same-bank-access stall
- Reserved port for future prefetcher implementation, 1-bit NRU replacing policy
Relative Performance & Progress Projection

Phase 1 (Now)
- Performance Modeling & Simulation
- Initial implementation
  - Porting existed L2 cache IP to Rocket chip
  - Adding Next Line to the latest Rocket chip
Relative Performance & Progress Projection

Phase 2 (Sept. 2018)

- Fully-matured Next-Line Prefetcher ( +10% IPC )
- Merging L2 Cache ( +10% IPC )
Relative Performance & Progress Projection

Phase 3 (Nov. 2018)

- Fully-matured IP Stride Prefetcher ( +10% IPC )
- Adding Aggressive L2 Prefetcher ( +10% IPC )
Relative Performance & Progress Projection

Phase 4 (Dec. 2018)
- Aggressive L2 Replacing Policy ( +5% IPC )
- Other architecture tuning ( +5% IPC)
Thanks!

- We can't make absolute promise, but will try hard to reach it.
- Our implementation will be kept opensourced with Labeled RISC-V branch
  https://github.com/LvNA-system/labeled-RISC-V
- Also welcome volunteers
- Testing more benchmarks
- Merging with master branch or checkout a new dedicated branch